Push Job Specification

Christopher Brown <cb@opscode.com>
Kevin Smith <kevin@opscode.com>

April 9, 2012

1 Overview

This specification describes the ability to execute on-demand chef-client via knife. For brevity's sake this feature will be referred to as "push job" or "push jobs".
The concept of push jobs is quite simple. A user is able to select some subset of nodes managed by their Private Chef organization, specify an action or command for those nodes to execute, and track the status of each node as it executes the request.
A bit of complexity and bookkeeping lurks underneath push job's simplicity. Managed nodes will need to a way to reliably monitor the job coordinators so they can cope with outages. Job coordinators also need to monitor the status of each managed node so commands are only sent to available nodes and to track job progress.
The remainder of this document will attempt to describe this feature in enough detail to allow a reasonable and scalable implementation.

2 Assumptions

2.1 Connectivity

1. Managed nodes **MUST** be reachable via a TCP-enabled network interface. 2. Managed nodes **MUST** be able to accept incoming TCP connections. 3. Managed nodes **MUST** be able to connect to the heartbeat and job coordination components inside Chef server.

2.2 Data format & Storage

1. All messages will be formatted as legal JSON. 2. The database is the canonical store of all application data.

2.3 Scalability & Security

1. Push jobs will be deployed in Private Chef only. 2. Push jobs will not be deployed in Hosted Chef. 3. The design must scale up to 8,000 managed nodes per OPC server. 4. Push jobs will honor the same security guarantees made by the Chef REST API.

3 Architecture

3.1 Communications

Managed nodes and server components will communicate using ZeroMQ messaging. Communication can be separated into two categories: heartbeat and job execution. The heartbeat channel is used by the Chef server to detect when managed nodes are offline and, therefore, unavailable for job execution. Managed nodes also use the heartbeat channel to detect when the server is unavailable.
The job execution channel is used by the Chef server to send job execution requests to managed nodes. Managed nodes used the execution channel to send job-related messages such as acknowledging jobs, sending progress updates, and reporting final results.

3.1.1 Heartbeat Channel

PUB and SUB sockets are used because they automatically and scalably manage the fanin the server requires to monitor nodes as well as the fanout the server needs to broadcast its heartbeat to all the nodes.

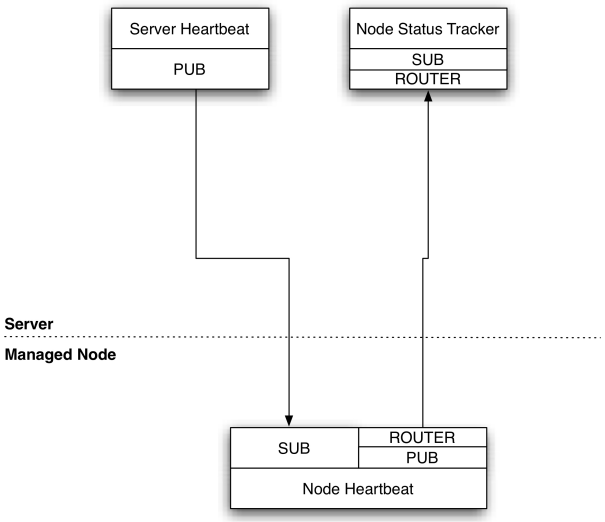

Figure 1: ZeroMQ sockets

The details of how nodes and servers discover and connect to each other's PUB and SUB sockets is covered in Server and Client Discovery.

3.1.2 Command Channel

TBD

3.2 General Messaging

3.2.1 JSON

Push jobs use JSON because ZeroMQ handles packet fragmentation and reassembly. JSON also facilitates easier debugging and maintenance of the system since all messages are textual and human-readable. A binary protocol, such as Protocol Buffers or msgpack, would be more

efficient but would also substantially increase the effort required to debug and support the system.

3.2.2 Security

All messages are signed using the caller's private key. This signature is transmitted in a separate ZeroMQ frame before the JSON payload.[1]

```
Sock = connect_to_server("tcp://some_server:8765"),
Sig = sign_message(JSON),
erlzmq:send(Sock, Sig, [sndmore]),
erlzmq:send(Sock, JSON)
```

3.2.3 Heartbeat

```
{
  "type":  "heartbeat",
  "host":  "node123.foo.com",
}
```

type message type
host the sender's hostname

3.3 Protocols

3.3.1 Heartbeat

Liveness detection in a distributed system is a notoriously difficult problem. The most common approach is to arrange for two parties to exchange heartbeat messages on a regular interval. Let's call these two parties 'A' and 'B'. Both A and B are considered 'online' while they are able to exchange heartbeat messages. If A fails to receive heartbeats from B for some number of consecutive intervals then A will consider B 'offline' and not route any traffic to B. A will update B's status to 'online' once A starts receiving heartbeats from B again.
This is the scheme used by this design. The Private Chef server sends out regular heartbeats to managed nodes via ZeroMQ PubSub. Managed nodes send their heartbeats over the command channel using the node status tracker's ZeroMQ identity. The identity is required so the message is correctly routed. See the Heartbeat Channel section for a visual representation of the message flows and ZeroMQ sockets.

[1] Public key signatures are used to verify the sender's identity and provide some amount of message tamper detection.

- Managed nodes
  A managed node sends heartbeats to the in_addr URL specified in the config data received during discovery. It receives server heartbeats via the out_addr URL specified during discovery. ZeroMQ Pub/Sub is strongly recommended for broadcasting server heartbeats since it's more efficient when sending the same message to many receivers. Clients should not ACK server heartbeats.
  A managed node must mark the OPC server as offline when it fails to receive server heartbeats for a consecutive number of intervals equal to push_jobs/heartbeat/offline_threshold. A managed client must not attempt to send any data when the server is offline. Any job requests received by the managed node from the offline server must be NAK'd and discarded.
  After a managed node has marked the server as offline it must receive server heartbeats for a consecutive number of intervals equal to push_jobs/heartbeat/online_threshold before marking the server online. The managed node may resume sending data and accepting job requests from the OPC server at this point.
- OPC Server
  A OPC server broadcasts its heartbeat on the URL described by out_addr. The server shouldn't expect any ACKs from managed nodes.
  The server receives heartbeats messages from managed nodes via ZeroMQ Pub/Sub. Each time a new managed node connects to a OPC server it tells the server the URL of its heartbeat publisher so it can subscribe to the managed node's heartbeat stream.
- **TODO** Do clients store and forward or just drop data when the server is unavailable?
- **TODO** How do we expose HWM/SWAP as user-visible tunables? Do we want to?

3.4 Server and Client Discovery

3.4.1 REST endpoint (perhaps on /nodes?) to supply all config data in JSON format

```
{
  "type": "config",
  "host": "opc1.opscode.com",
  "push_jobs": {
                "heartbeat": {
                              "out_addr": "tcp://10.10.1.5:10000",
                              "in_addr": "tcp://10.10.1.5:10001",
                              "interval": 15,
                              "offline_threshold": 3,
                              "online_threshold": 2
                            },
                "command":   {
                              "addr": "tcp://10.10.1.5:10001"
                            }
```

```
              },
  "public_key": "AAAAB3NzaC1kc3MAAACBAIZbwlySffbB
                 5msSUH8JzLLXo/v03JBCWr13fVTjWYpc
                 cdbi/xL3IK/Jw8Rm3bGhnpwCAqBtsLvZ
                 OcqXrc2XuKBYjiKWzigBMC7wC9dUDGwDl
                 2aZ89B0jn2QPRWZuCAkxm6sKpefu++VPR
                 RZF+iyZqFwS0wVKtl97T0gwWlzAJYpAAA
                 AFQDIipDNo83e8RRp7Fits0DSy0DCpwAA
                 AIB01BwXg9WSfU0mwzz/0+5Gb/TMAxfkD
                 yucbcpJNncpRtr9Jb+9GjeZIbqkBQAqwg
                 dbEjviRbUAuSawNSCdtnMgWD2NXkBKEde",
  "sig":   "yYy96kyXcpV840fCW7O2Nw=="

}
```

type message type
host sender's host name (Private Chef server)
push_jobs/heartbeat/out_addr URL pointing to the server's heartbeat broadcast service
push_jobs/heartbeat/in_addr URL pointing to the server's node state tracking service
push_jobs/interval Interval, in seconds, between heartbeat messages
push_jobs/offline_threshold How many intervals must be missed before the other end is considered offline
command/addr URL pointing to the command service
sig Base64 encoded cryptographic signature of the stringified JSON hash containing only the above fields[1]